Marist College

MS in Computer Science
School of Computer Science and Mathematics

CMPT-435L-111-23s Algorithm Analysis and Design
Brian Gormanly
Spring 2023

Assignment 1: Data Structures

Connor H. Johnson
connor.johnson1@marist.edu

Assignment 1: Data Structures

Connor H. Johnson
connor.johnson1@marist.edu

February 20, 2023

Overview

This document will cover how I created a node, stack, and queue inside a java file that successfully depicts whether a given string is a palindrome. I will also include the following:

- The code I used to complete this project.
- Short explanations on certain parts of the code.
- Resources to look at for reference.

Code listings

First, I would like to show you the two files I used to create this project: `mainProgram.java` and `singlyLinkedList.java`.

The code provided will not consist of comments for visualization purposes; if you want to see my comments in my code, please visit the GitHub repository[1]

1 mainProgram.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import singlyLinkedList.singlyLinkedList;
public class mainProgram {

   public static void main(String[] args) {
      try {
          File file = new File("/Users/Johnson_code/CJohnson-435/CJohnson-
                                    435/Lab1/textFiles/magicitems.txt");
          Scanner scanner = new Scanner(file);

          while (scanner.hasNextLine()) {
```

[1] GitHub: https://github.com/MaristGormanly/CJohnson-435/tree/main/Lab1

```
               String line = scanner.nextLine();
               String originalLine = line;
               line = line.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
               singlyLinkedList sStack = new singlyLinkedList();
               singlyLinkedList sQueue = new singlyLinkedList();

               for (char c : line.toCharArray()) {
                   sStack.push(c);
                   sQueue.enqueue(c);
               }

               boolean truePal = true;
               while (!sStack.isEmpty() && !sQueue.isEmpty()) {
                   char stackChar = (char) sStack.pop();
                   char queueChar = (char) sQueue.dequeue();
                   if (stackChar != queueChar) {
                       truePal = false;
                       break;
                   }
               }

               if (truePal) {
                   System.out.println(originalLine);
               }
           }
           scanner.close();
       } catch (FileNotFoundException e) {
           e.printStackTrace();
       }
   }

}
```

2 singlyLinkedList.java

```
package singlyLinkedList;
public class singlyLinkedList {

   class Node{
       char data;
       Node next;

       public Node(char data) {
           this.data = data;
           this.next = null;
      }
  }

   public Node head = null;
```

```
    public Node tail = null;

    public void addNode(char data) {
        Node newNode = new Node(data);

        if(head == null) {
            head = newNode;
            tail = newNode;
        }
        else {
            tail.next = newNode;
            tail = newNode;
        }
    }

    public void addFront(char data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            newNode.next = head;
            head = newNode;
        }
    }

    public void addEnd(char data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            tail = newNode;
        }
    }

    public void removeFront() {
        if (head == null) {
            return;
        }

        head = head.next;

        if (head == null) {
            tail = null;
        }
    }

    public void removeEnd() {
        if (head == null) {
            return;
        }

        if (head == tail) {
            head = null;
            tail = null;
        } else {
            Node current = head;
            while (current.next != tail) {
                current = current.next;
            }
            current.next = null;
            tail = current;
        }
    }

    public void print() {
        Node current = head;

        if(head == null) {
            System.out.println("List is empty");
            return;
        }
        System.out.println("Nodes of singly linked list: ");
        while(current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public void push(char data){
         Node newNode = new Node(data);

         if (head == null) {
             head = newNode;
            tail = newNode;
         } else {
             newNode.next = head;
             head = newNode;
         }
     }

     public char pop() {
         if (head == null) {
             return 0;
         }

         char data = head.data;
         head = head.next;

         if (head == null) {
             tail = null;
         }

         return data;
     }


     public void printStack() {
         Node current = head;

         if (head == null) {
             System.out.println("Stack is empty");
             return;
         }

         System.out.println("Nodes of Stack singly linked list: ");

         while (current != null) {
             System.out.print(current.data + " ");
             current = current.next;
         }

         System.out.println();
     }

     public void enqueue(char data){
         Node newNode = new Node(data);

         if (head == null) {
             head = newNode;
             tail = newNode;
         } else {
             tail.next = newNode;
             tail = newNode;
         }
     }

     public char dequeue() {
         if (head == null) {
             return 0;
         }

         char data = head.data;
         head = head.next;

         if (head == null) {
             tail = null;
         }

         return data;
     }

     public void printQueue(){
         Node current = head;

         if(head == null) {
             System.out.println("List is empty");
             return;
         }
         System.out.println("Nodes of Queded singly linked list: ");
         while(current != null) {
             System.out.print(current.data + " ");
             current = current.next;
         }
         System.out.println();
     }

     public int length(Node head){
         int count = 0;
         Node current = head;
         while(current != null){
             count++;
             current = current.next;
         }
         return count;
     }

     public boolean isEmpty() {
         return head == null;
     }
 }
```

Code Explanation

Now that both files can be reviewed, it is important to review parts of the code essential to completing this project. Below will be breakdowns of key components of the `singlyLinkedList` methods being used inside of `mainProgram.java`.

1.) Class Node (singlyLinkedList.java)

- Inside of singlyLinkedList.java, we start out by initializing the Node head and the tail of our singly linked list (Lines: 4-15):

  For reference on where I found the class Node code, please visit the site linked[2]

```
        class Node{
        char data;
        Node next;

        public Node(char data) {
            this.data = data;
            this.next = null;
        }
    }

    public Node head = null;
    public Node tail = null;
```

  The importance of initializing our Node head and tail is because we need to keep track of the head and tail of the list to add and remove nodes from the list. Without this feature, key methods like .push(), .pop(), .enqueue(), or .dequeue() could never work.

2.) push() method (singlyLinkedList.java)

- Inside of singlyLinkedList.javasection, we construct the Push() method (Lines: 99-109):

```
    public void push(char data){
        // Create a new node with the given data
        Node newNode = new Node(data);

        // If the list is empty, set head and tail to the new node
        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            // set the new nodes next to the current head
            newNode.next = head;
            // Set the head to the new node
            head = newNode;
        }
    }
```

- the push() method is simply adding an element to the top of a stack. Inside of the mainProgram.java inside our loop, we read characters one by one to get their indexes. From that, we can push those single characters to a stack and obtain individual characters of a string.

[2] JavaTPoint: https://www.javatpoint.com/java-program-to-create-and-display-a-singly-linked-list

3.) pop() method (singlyLinkedList.java)

- Inside of singlyLinkedList.java section, we construct the Push() method (Lines: 111-114):

```
public char pop() {
    // If the list is empty, do nothing
    if (head == null) {
        return 0;
    }

    // remove the current head
    char data = head.data;
    head = head.next;

    // If the head is null, set the tail to null as well
    if (head == null) {
        tail = null;
    }

    return data;
}
```

- the pop() method is key when trying to remove an element from the top of a stack. In this case, we are utilizing the pop method to remove characters from the stack and then making sure the character from the queue is the same

4.) enqueue() method (singlyLinkedList.java)

- Inside of singlyLinkedList.java section, we construct the Push() method (Lines: 145-155):

```
public void enqueue(char data){
    // Create a new node with the given data
    Node newNode = new Node(data);

    // If the list is empty, set head and tail to the new node
    if (head == null) {
        head = newNode;
        tail = newNode;
    } else {
        // set the current tail's next to the new node
        tail.next = newNode;
        // Set the tail to the new node
        tail = newNode;
    }
}
```

- the `enqueue()` method almost does the same as the `push()` method; however, now it will add an item to the back of the queue. This is really helpful when trying to locate palindromes because when you have a stack reading it forwards, you now have a queue reading the strings backward, making it easy to check if the string will be a palindrome

4.) dequeue() method (`singlyLinkedList.java`)

- Inside of `singlyLinkedList.java` section, we construct the `Push()` method (Lines: 157-170):

```
public char dequeue() {
    // If the list is empty, do nothing
    if (head == null) {
        return 0;
    }

    // remove the current head
    char data = head.data;
    head = head.next;

    // If the head is null, set the tail to null as well
    if (head == null) {
        tail = null;
    }

    return data;
}
```

- the `dequeue()` method almost does the same as the `pop()` method; however, now it removes an item from the front of the queue. This and the `pop()` method is going through their respective character-based strings and checking to see if they match in characters, making it a palindrome.

  Based on these four methods, You can now start to visualize the process inside of `mainProgram.java` when checking for palindromes; however, there are still some unused methods left in our node, stack, and queue class, like:

  – addNode() : adds a single character to a singly linked list
  – addFront() : adds a single character to the front of a singly linked list
  – addEnd() : adds a single character to the end of a singly linked list
  – removeFront() : removes a single character at the front of a singly linked list
  – removeEnd() : removes a single character at the end of a singly linked list
  – length() : iterates and counts through a string to find the length of the string

5.) Palindrome Checker (`mainProgram.java`)

Inside of `mainProgram.java`, we create the necessary code to make a Palindrome checker (Lines: 1-45):

- Inside of this, we are reading a specific file to 'scanning' or reading the file so that it is callable inside of our code (Lines: 10-11):

```
File file = new File("/Users/Johnson_code/CJohnson-435/CJohnson-435
                        /Lab1/textFiles/magicitems.txt");
Scanner scanner = new Scanner(file);
```

- After we make a while loop that scans through all the lines in the .txt file found with the scanner. In Java, there is no need to make an exception inside of the while loop because once scanner.hasNextLine() does not have a next line, the loop will end. From then, we initialize our scanner.nextLine() function and now store the original string line and then store a separate string line to remove all non-alphanumeric characters and convert them to lowercase. After we initialize our singly linked list to create our stack and queue.(Lines: 10-11):

```
while (scanner.hasNextLine()) {
    String line = scanner.nextLine();
    String originalLine = line;

    line = line.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();
    singlyLinkedList sStack = new singlyLinkedList();
    singlyLinkedList sQueue = new singlyLinkedList();
```

- Once this is done, we than me a common loop to iterate through a string to store individual characters, which is done here and stored into a stack and queue. The reason we do both is that the stack will store the character from the front on (Example: connor), but the queue will reverse store it (Example: ronnoc)(Lines: 20-23):

```
for (char c : line.toCharArray()) {
    sStack.push(c);
    sQueue.enqueue(c);
}
```

- The final step to see if the string will be a palindrome is to remove the characters from the stack and queue at the same time and check to see if they are the same characters. This also removes the characters so that the stack and queue are going to be empty for the line run by the scanner. With truePal being set to true, the loop will end if the characters in the stack and queue are different and cause the while loop to end due to truePal being false, but if it remains true for the whole loop, then it is considered a palindrome and printed out.(Lines: 25-38):

```
    boolean truePal = true;
    while (!sStack.isEmpty() && !sQueue.isEmpty()) {
        char stackChar = (char) sStack.pop();
        char queueChar = (char) sQueue.dequeue();
        if (stackChar != queueChar) {
            truePal = false;
            break;
        }
    }

    if (truePal){
        System.out.println(orginialLine);
    }
```

- This last part is a small but good practice while coding. on line 39, I close the scanner in good practice and end my try expression with a catch to throw an error to me in case it can not find the .txt file(Lines: 39-45):

```
    scanner.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
```

Resources Used

Here is a list of resources I used throughout my completion of this project:

Creating my Node class:

- https://www.javatpoint.com/java-program-to-create-and-display-a-singly-linked-list

Checking if a string is a palindrome:

- https://www.geeksforgeeks.org/function-to-check-if-a-singly-linked-list-is-palindrome/

linked lists:

- https://www.w3schools.com/java/java$_l$inkedlist.asp
- https://youtu.be/YQQio9BGWgs
- https://www.programiz.com/dsa/linked-list

Length method:

- https://www.youtube.com/watch?v=krLRbqAV6wI

Methods:

- https://www.youtube.com/watch?v=ILJgewz5Dxwt=30s
- https://www.youtube.com/watch?v=91CMnJeHJVct=389s
- https://www.geeksforgeeks.org/adding-an-element-to-the-front-of-linkedlist-in-java/
- https://www.programiz.com/dsa/stack

Connecting java files

- https://www.youtube.com/watch?v=3ybNZM6cP3M

Debugging

- https://openai.com/blog/chatgpt/

Helping with visualiztion

- https://pythontutor.com/visualize.htmlmode=display